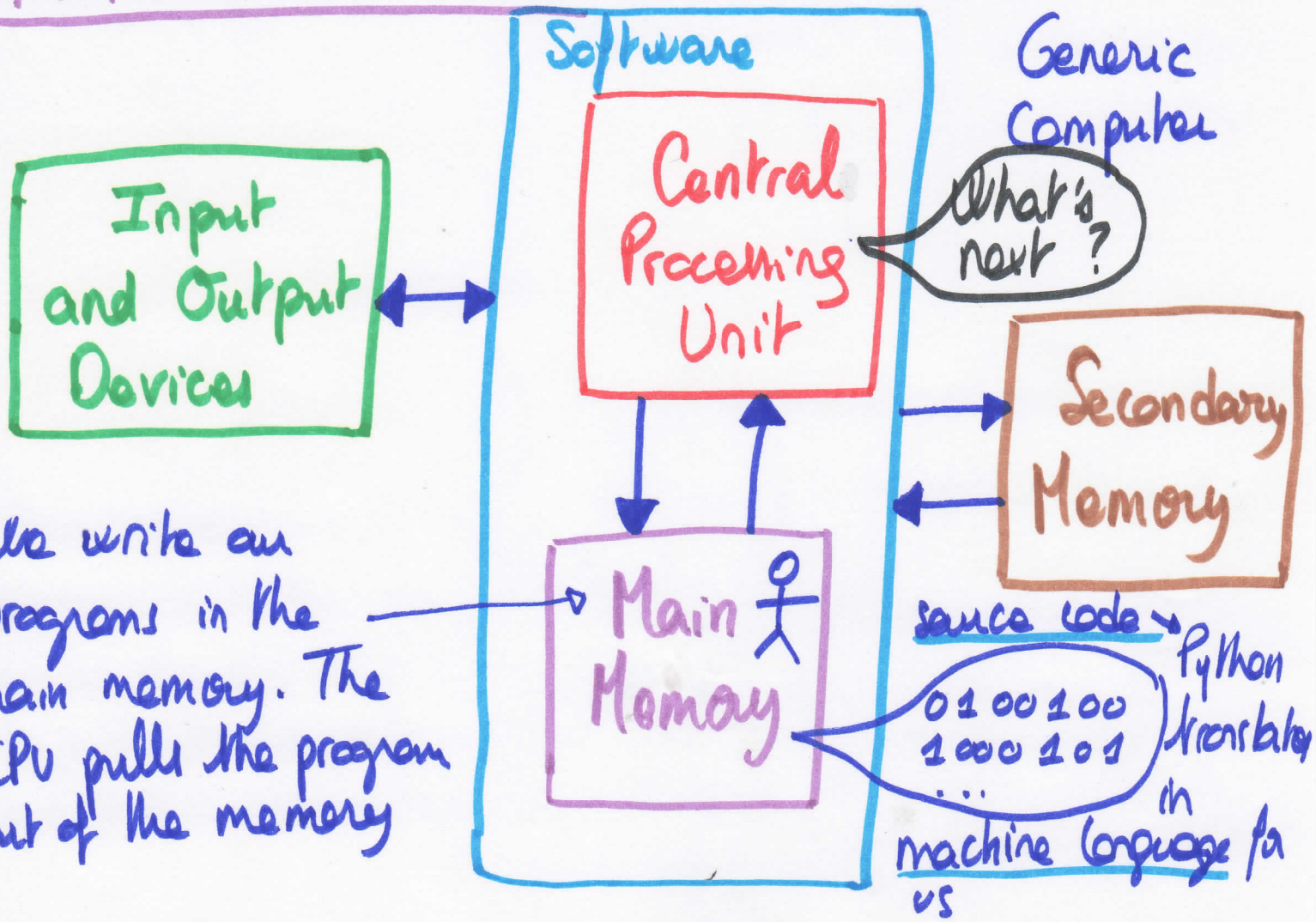
① Programming for everybody (Python) Uni of Michigan Week 1

Hardware architecture



Central Processing Unit: Runs the program. The CPU is always wondering "what to do next"? Closer to a brain, but dumb and fast

Input Devices: Keyboard, Mouse, Touch screen

Output Devices: Screen, Speakers, Printer, DVD Burner

Main Memory: Fast small temporary storage - lost on reboot - aka RAM (Needs power)

Secondary memory: Slower large permanent storage - lasts until deleted - disk drive / memory stick

② Programming for everybody (Python)

Univ of Michigan

week 4

Source code → Python program → Machine language

Constants

Fixed values (numbers, letters, string). Their value does not change. String constants use simple quotes (') or double quotes (")

Variables

Named place in the memory where a programmer can store data and later retrieve the data using the variable "name". You choose the name and can change the content of a variable in a later statement

Python variable name rules

1- Must start with a letter or underscore
2- Must consist of letters and numbers and underscore
3- Case sensitive (upper-lower case)

Reserved words: if, for, while, class, print...
You cannot use them as variable names / identifiers.

Numeric expressions

Operator	+	-	*	/	**	%
Operation	Addition	Substract.	Mult.	Division	Power	Remainder

③ Programming for everybody (Python) Uni of Michigan Week 4

Operator precedence rules

① Parenthesis
② Power (Exponentiation)
③ Multiplication, Division and Remainder
④ Addition and Substraction
⑤ Left to Right

Integer and floating division

- Integer division truncates. print 10/2 5
- floating point division produces floating point numbers print 10.0/2.0 5.0
- Integer & floating point operands result in a floating point number print 99/100.0 0.99

Type

type() . Floating point numbers have decimal part: 99.6..

Type conversion: int() float()

User Input

raw_input() pauses and reads data from the user. It rehurns a string

```
nam = raw_input('Who are you?')
print 'Welcome', name
```

output

```
Who are you? Steph
Welcome Steph
```

④ Programming for everybody (Python) Univ. Michigan Week 4-5

Comments

String operations + concatenation * multiple concatenation. e.g. print 'Hi'*5 HiHiHiHiHi

Comparison operators

Python	<	<=	==	>=	>	!=
Meaning	Less than	less than or equal	Equal to	Greater than or equal	Greater than	Not equal

'=' is used for assignment

Boolean expression

Ask a question and produce a Yes or No result which we use to control program flow.
Boolean expressions using comparison operators evaluate to True / False - Yes / No

Indentation

- Increase indent after an if or for statement
- Maintain indent to indicate the scope of the block (which lines are affected by the if / for)
- Reduce indent back to the level of the if statement or for statement to indicate the end of the block

Blank lines are ignored. They do not affect indentation

⑤ Programming for everybody (Python) Univ of Michigan week 5

Comments on a line by themselves are ignored with regard to indentation.

Multiway decisions

```
x = 0
if x < 2 :
    print 'small'
elif x < 10:
    print 'medium'
else :
    print 'LARGE'
print 'All done'
```

1 if
2 elif
3 else

Try/except structure

It is used to surround a dangerous section of code

- if the code in the try works - the except is skipped
- if the code in the try fails - it jumps to the except section

Example

```
astr = 'Hello Bob'
try:
    istr = int(astr)
except:
    istr = -1
print 'First', istr
x = 10
print x
```

When the first conversion fails, it just drops into the except: clause and the program continues

output:

```
First -1
10
```

⑥ Programming for everybody (Python)

Univ of Michigan
week 6-7

Python functions

- Built-in functions: provided as part of Python. max()...
- Functions that we define ourselves

Example

parameters

function definition (stores function, doesn't print) ← def addtwo(a, b)

```
    added = a + b
    return added
x = addtwo(3, 5)
print x
8
```

arguments

return a value to be used as the value of the function

Example

```
def greet()
    return "Hello"
print greet(), "Steph"
```

output: Hello Steph

Void functions (non-fruitful):

When a function does not return a value

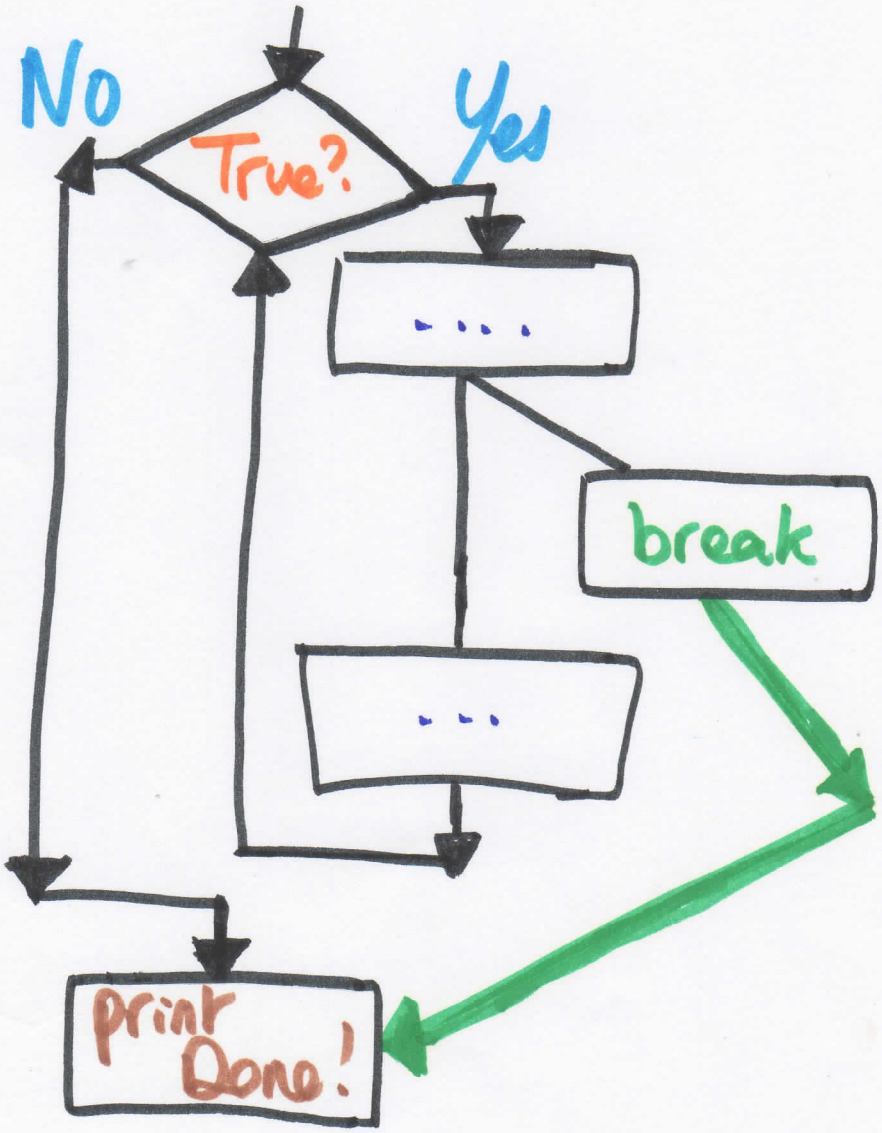
Indefinite loops (while)

They keep going until a logical condition becomes False. Make sure the loop terminates, careful with infinite loops

Breaking out of a loop

The **break** statement ends the current loop and jumps to the statement immediately following the loop. It can be anywhere in the body of the loop.

```
while True:
    line = raw_input('> ')
    if line == 'done':
        break
    print line
print 'Done!'
```



output example:

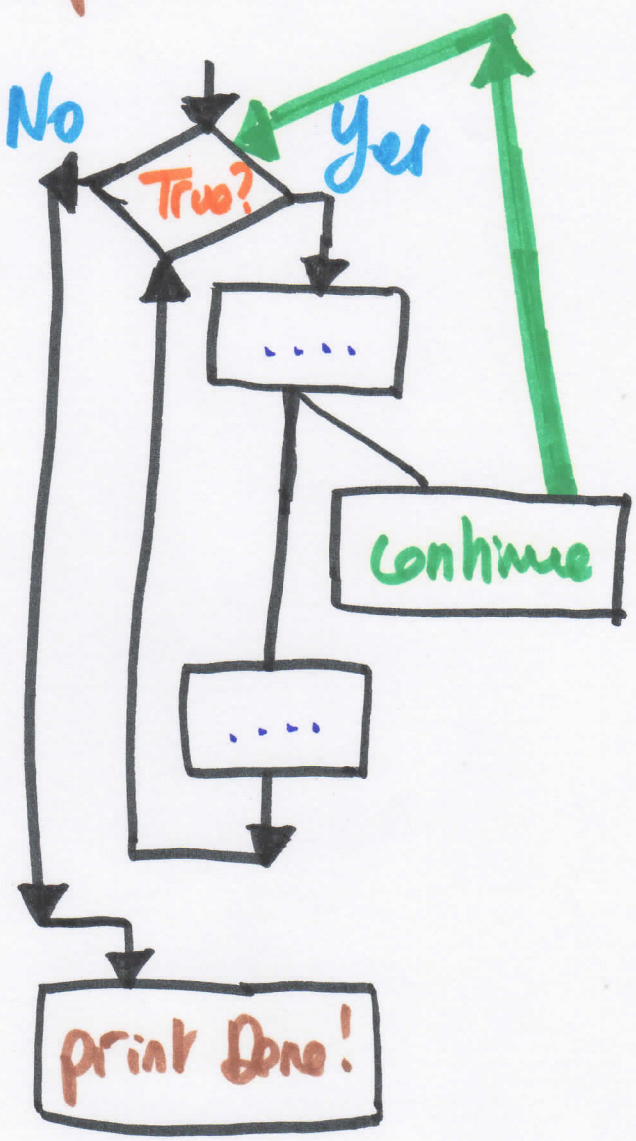
```
> hello there
hello there
> finished
finished
> done
Done!
```

⑧ Programming for everybody (Python) Uni of Michigan Week 7

Finishing an iteration with continue

The **continue** statement ends the current iteration and jumps to the top of the loop and start the next iteration.

```
while True
    line = raw_input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print line
print 'Done!'
```



Definite loops (for)

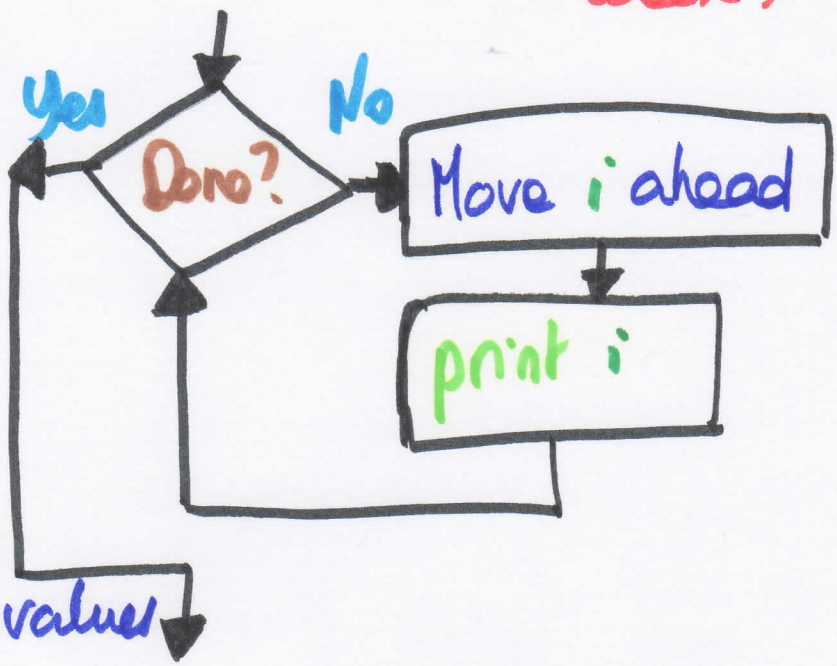
They iterate through the members of a set. They execute an exact number of times.

```
for i in [5,4,3,2,1]:
    print i
print 'Blastoff!'
```

- `i` — iteration variable
- `[5,4,3,2,1]` — Five element sequence
- `print i` — block (body)

⑨ Programming for everybody (Python) Uni of Michigan week 7

- The iteration variable "iterates" through the sequence
- The body of code is executed once for each value in the sequence
- The iteration variable moves through all of the values in the sequence.



The 'is' and 'is not' operators

is and is not are logical operators. is implies "is the same as", similar to, but stronger than == Good to use it in some cases but careful. Sometimes better == (when numbers...)

Finding the smallest value

```
smallest = None
print 'Before'
for value in [9,41,12,3,74,15]:
    if smallest is None:
        smallest = value
    elif value < smallest:
        smallest = value
    print smallest, value
print 'After', smallest
```

output

```
Before
9 9
9 41
9 3
3 3
3 74
3 15
After 3
```